Technische Hochschule Ingolstadt

Seminararbeit/Whitepaper

Umgehen von USB-Deskriptor basierten USB-Policies am Beispiel einer virtuellen Umgebung

angefertigt von
Dominik Schlecht

Betreuer:

Technische Hochschule Ingolstadt:	Dr. Stanislaus
Allianz Deutschland AG:	Dr. Stremmel und Herr Gerhager

Ingolstadt, 1. November 2014

Sperrvermerk

Die vorliegende Arbeit „Umgehen von USB-Deskriptor basierten USB-Policies am Beispiel einer virtuellen Umgebung“ wurde mit Unterstützung der Allianz Deutschland AG erstellt. Es sind jedoch keine internen, vertraulichen oder streng vertraulichen Informationen enthalten. Die Weitergabe des Inhalts der Arbeit im Gesamten oder in Teilen sowie das Anfertigen von Kopien oder Abschriften – auch in digitaler Form – sind grundsätzlich erlaubt.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Seminararbeit bis auf die offizielle Betreuung durch den Aufgabensteller selbstständig und ohne fremde Hilfe verfasst habe.

Die verwendeten Quellen sowie die verwendeten Hilfsmittel sind vollständig angegeben. Wörtlich übernommene Textteile und übernommene Bilder und Zeichnungen sind in jedem Einzelfall kenntlich gemacht.

Ingolstadt, 1. November 2014

Danksagung

An dieser Stelle möchte ich mich bei allen Bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben. Besonderer Dank dabei gebührt den Kollegen aus der Allianz Deutschland AG und meinen Eltern.

Dominik Schlecht, 1. November 2014

Inhaltsverzeichnis

1. Einleitung

In Zeiten von *Heartbleed* [2] und *Shellshock* [5], *Snowden* und der *NSA* [6] und der fortlaufenden Digitalisierung der Industrie und Gesellschaft wird das Thema Informationssicherheit immer wichtiger. Daten werden, unabhängig davon, ob diese Privatpersonen oder Unternehmen zugeordnet sind, immer wertvoller. So ergeben sich beispielsweise aus einem gehackten Smartphone einer Privatperson Informationen wie E-Mail-Adressen, Kontakte und Chat-Verläufe bis hin zu Passwörter für Online-Banking oder persönlichen Bildern. Wenn diese Informationen auf dem Schwarzmarkt verkauft oder online veröffentlicht werden, kann dies für die Personen oft Reputation wie auch finanzielle Schäden nach sich ziehen. Diese Tätigkeiten werden oft unter dem Schlagwort Cybercrime zusammengefasst. Betrachtet man Unternehmen, so ist der möglichen finanziellen Schaden wesentlich höher als für Privatpersonen. Durch die Entwendung von Kreditkarten erlitten zum Beispiel mehrere Supermärkte in den USA beträchtliche Reputationsschäden [1] [3]. Eventuell noch höhere Schäden könnte es nach sich ziehen, wenn streng vertrauliche Dokumente von Unternehmen, wie z.B. Konstruktionsskizzen für ein neues Automodell, Quellcode oder vorläufige Geschäftsberichte, durch Hacker erbeutet und an ein Konkurrenzunternehmen verkauft würden. Dies hört sich irreal an, aber die die Firma McAffee schätzt den Verlust für die Wirtschaft durch „Cybercrime“ im Jahr 2014 auf bis zu 575 Milliarden USD [4]. Um diesem Trend entgegen zu wirken, müssen Unternehmen Maßnahmen ergreifen, welche das Schutzniveau erhöhen. Oft werden hier auf der technischen Seite nur internetseitige Komponenten beachtet, wie das schnelle Patching von Servern. Dies ist in Hinsicht auf Poodle und Shellshock sicherlich auch notwendig, jedoch sollte man alle Wege, über welche Daten von Dritten in das Unternehmen gelangen, Daten an Dritte weitergeben werden könnten, sowie auch interne Bedrohungen wahrnehmen, einschätzen und eindämmen. Eine solche Prüfung war die Grundlage für dieses Dokument.

2. Szenario

Bei einer Prüfung interner Regularien bei der Informationssicherheit wurde das Thema USB-Geräte in Verbindung mit Thin oder auch sogenannten Zero-Clients aufgegriffen. Hier soll aus gegeben fachlichen Anlässen eine Möglichkeit geschaffen werden, lokale USB-Geräte, wie z.B. USB-Sticks oder USB-CD-Laufwerke, an die virtuelle Maschine des Benutzers weiter zu leiten. Hier galt es das Risiko zu prüfen und entsprechende Gegenmaßnahmen zu entwickeln. Würde man das Durchstellen von USB-Geräten im Allgemeinen erlauben, so würden sich erheblich Gefahren ergeben, welche unter 3.1 erläutert werden. Um dem vorzubeugen, soll auf Basis einer Policy, welche in 4 weiter erläutert werden, das Durchstellen auf bestimmte Geräte begrenzt werden. Dies geschieht bei dem hier getesteten Produkt über die Filterung nach USB-Deskriptoren wie *idVendor* und *idProduct*, welche ein USB-Device, z.B. einen bestimmten USB-Stick oder ein USB-CD-Laufwerk, eindeutig identifizieren sollen. Diese Felder werden unter 3.2 weiter erläutert. Bei der durchgeführten Sicherheitsprüfung stellte sich jedoch heraus, dass USB-Deskriptoren keinen Sicherung unterliegen und somit mit bestimmten Geräten gezielt Emuliert werden können. Diese Umgehung der Policy soll in diesem Dokument erläutert und aufgezeigt werden. Den Proof-Of-Concept finden Sie unter 5.4.

Schritt 1: Ein USB-Gerät wird angesteckt.

Schritt 2: Der USB-Treiber des Thinclients bindet das Gerät ein. TODO Prüfen was übertragen wird

Schritt 3: Der Thinclient leitet entsprechende Deskriptor-Felder an den Hypervisor weiter.

Schritt 4: Der Hypervisor prüft die Deskriptor-Felder gegen die Hypervisor-Policy. Diese erlaubt entweder das Durchstellen oder verbietet es. Wird die Durchstellung verboten, wird der USB-Stick nicht an die Hypervisor-Umgebung weitergeleitet und damit würde der Ablauf enden. Im Folgenden wird angenommen, dass der USB-Stick weitergeleitet wird.

Schritt 5&6: Der Hypervisor fordert den USB-Stick beim Thinclient an und bindet diesen ein.

Schritt 7: Der Hypervisor gibt das USB-Device an die VM weiter.

Schritt 8: Die VM prüft das USB-Device anhand der Deskriptor-Felder gegen die VM-Policy und bindet diesen entweder ein oder lehnt diesen ab.

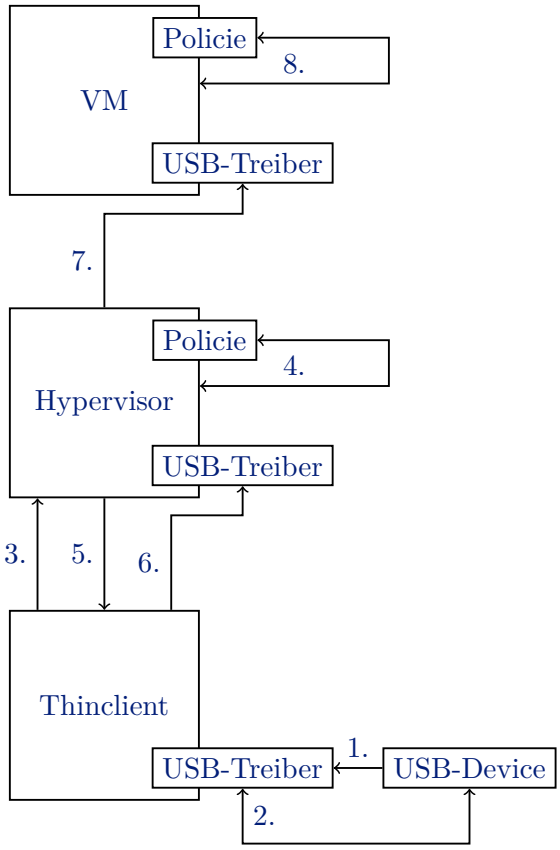

Abbildung 2.1.: Ablaufübersicht

3. USB

USB ist eine Schnittstelle, welche so gut wie alle modernen Rechner besitzen. Es ist unter anderem möglich darüber Geräte wie Kopfhörer, Joysticks aber auch Wechseldatenträger mit dem Rechner zu verbinden. Im letzteren Bereich ersetzt USB die bisher vorherrschende CD/DVD-Technologie, da auf einem USB Stick mehr Daten in höherer Geschwindigkeit gespeichert werden können und zudem handlicher sind. Jedoch birgt die USB-Technologie auch Gefahren, welche in den folgenden Sektionen beschrieben werden. Anschließend werden in 3.2 die Deskriptoren beschrieben, welche ein USB-Gerät mit sich bringt und für die Prüfung gegen die Policy eine besondere Rolle spielen.

3.1. Gefahren bei USB

USB-Geräte stellen Gefahren auf verschiedenen Ebenen dar. Zum einen werden USB-Sticks, zumindest in dem Szenario, das hier betrachtet wurde, von Dritten an Mitarbeiter gegeben. Das bedeutet, dass ein Dritter, insofern dieser die nötige kriminelle Energie aufweist, ein präpariertes Gerät einschicken könnte. Erschwerend kommt hinzu, dass der Mitarbeiter keine Möglichkeit hat, ein böses USB-Gerät von einem normalen zu unterscheiden. Im folgenden werden einige der Risiken dargestellt

3.1.1. Viren

Viren sind eine wachsende Bedrohung in der heutigen Zeit. Vor einigen Jahren waren einige wenige Virenfamilien weit verbreitet. So konnten Virenschutzhersteller über signaturbasierte Suchalgorithmen nach bekannten Mustern suchen und Viren identifizieren. In den letzten Jahren zeichnet sich jedoch der Trend ab, dass Viren sich schneller weiterentwickeln und zudem oft polymorph programmiert sind, also ihr Aussehen bei einer Infektion verändern. Dadurch werden signaturbasierte Erkennungen immer ineffizienter und die Gefahr, dass ein Rechner unerkannt infiziert wird, steigt. Eine Infektion passiert zumeist über sogenannte Browserexploits, also präparierte Webseiten, welche Lücken in der Software des Users nutzten, oder Anhänge an Mails, welche Schadcode enthalten. In dem von hier betrachteten Szenario würde ein krimineller Dritter oder aber auch ein unwissender Dritter, dessen Rechner im Vorfeld von einem Virus infiziert wurde, einen USB-Stick mit einem Virus einschicken. Ein Beispiel für einen solchen Virus wäre ein Trojaner. Diese tarnen sich als normale Software, beinhalten aber auch Schadcodefunktionalität. [10] Will ein Benutzer das vermeidlich sinnvolle Programm installieren, wird im Hintergrund unbemerkt die Schadroutine mit installiert und gestartet. Dieser Schadecode hat oftmals Funktionalitäten wie Keylogger, Backdoors

oder ein Rootkit. Als Beispiel könnte man hier den in *Metasploit*[1] enthaltenen *Meterpreter* nennen, dessen Funktionen jedoch weit über die oben genannten hinaus gehen [2]. Hier muss also ein Benutzer einen USB-Stick einstecken und ein darauf befindliches Programm starten, damit sich der Virus installieren kann. Ist dieser bereits bekannt, könnte ein auf dem System installierter Virenscanner diesen finden und bestenfalls blockieren. Jedoch ist es aufgrund des technischen Fortschritts immer öfter der Fall, dass Viren trotz gleicher Funktionalität ihr aussehen selbst verändern können und dadurch von den Pattern des Virenschutzes nicht mehr erfasst werden.

3.1.2. Datenabfluss

Neben den Gefahren von außen müssen auch sogenannte „Inside-Threaths“ beachtet werden. Dies wären Mitarbeiter, welche z.B. interne IT-Systeme manipulieren, um sich Vorteile oder Reichtümer zu verschaffen. Bezogen auf USB wäre ein Risiko der Abfluss von vertraulichen oder wertvollen Daten, also wenn ein Mitarbeiter diese auf einem USB-Stick speichert und aus dem Machtbereich des Unternehmens bringt. Anschließend könnte er sich an diesen bereichern, falls diese Daten zum Beispiel Bankdaten umfassten. Andere denkbar Beispiele sind Kundendaten, Benutzerkontos, Geschäftsberichte oder sonstige Unternehmensgeheimnisse. Diese können oft für Geld in einschlägigen Bereichen des Internets verkauft oder bei Geschäftsberichten zur Manipulation am Finanzmarkt genutzt werden. Auch wäre eine Abwerbung eines Mitarbeiters von einem anderen Unternehmen für Industriespionage denkbar. Eine neue Bedrohung sind Geheimdienste, welche Personen in eine Unternehmen einschleusen oder Mitarbeiter abwerben, um Daten über die Kunden zu sammeln. Dies wurde erst letztens durch von Edward Snowden veröffentlichte Dokumente publik.[3]

3.1.3. Exploits auf Treiberebene

Für den Mitarbeiter noch schwieriger zu entdecken sind Exploits auf Treiberebene. Dieses Vorgehen ist relativ neu, es werden dabei Lücken im Treiber des Geräts ausgenutzt um Schadcode auszuführen. Hierzu muss ein Benutzer einen z.B. manipulierten USB-Stick nur einstecken. Es bedarf im Gegensatz zu normalen Viren keiner weiteren Interaktion des Users, da sich der Computer automatisch mit dem USB-Device kommuniziert um die Funktionen des Geräts zu erfahren und eventuell benötigte Treiber zu installieren. Hier beginnt das Gerät jedoch bereits bestimmte schädliche Zeichenfolgen an den Computer zu senden, welche vom Treiber interpretiert und unter Umständen einen Buffer Overflow oder eine andere Schwachstelle ausnutzen können[4]. Durch diese Lücken kann dann auf dem Rechner des Benutzers Schadcode ausgeführt werden, ohne das dieser dies bemerkt.

Dieser kurze Abschnitt gibt eine genauere, jedoch immer noch oberflächlich gehaltene Beschreibung der Schritte 2., 6. und 7. der Grafik 2.1. Wird ein USB-Gerät eingesteckt, bekommt dieser über den USB-Anschluss Strom und sendet ein Ankündigungspaket. Der

[1] http://www.metasploit.com/
[2] http://www.offensive-security.com/metasploit-unleashed/Meterpreter_Basics
[3] https://firstlook.org/theintercept/2014/10/10/core-secrets/
[4] https://srlabs.de/badusb/

Computer reagiert darauf und fordert Informationen, also die in 3.2 beschriebenen Felder, an. Das USB-Gerät überträgt diese an den PC. [7]

3.2. Deskriptoren

Der USB-Spezifikation, welche von dem USB Implementers Forum, Inc. [8] festgelegt wird, sieht Felder vor, welche Informationen zu dem Gerät beinhalten.

Bytes	
1	bNumConfigurations
1	iSerialNumber
1	iProduct
1	iManufacturer
2	bcdDevice
2	idProduct
2	idVendor
1	bMaxPacketSize
1	bDeviceProtocol
1	bDeviceSubClass
1	bDeviceClass
2	bcdUSB
1	bDescriptorType
1	bLength

Dies umfasst die technische Informationen wie die Länge der gesamten Felder im „bLength“-Feld oder das Protokoll des Geräts im „bDeviceProtocoll“-Feld über Informationen für das Betriebssystem wie „idVendor“, „idProduct“, „bDeviceSubClass“ und „bDeviceClass“. Diese Felder mit der jeweiligen Länge sind in der Grafik dargestellt. Ein Feld hat dabei zwischen ein und zwei Bytes. Die Felder „bDeviceClass“, „bDeviceSubClass“, „bDeviceProtocol“ sowie „idVendor“ werden vom Hersteller befüllt. [7] Das Betriebssystem nutzt die Felder meist um Treiber zu suchen oder auch das angeschlossene USB-Gerät gegen die Policy-Einstellungen zu prüfen. Um eigene Werte bei *idProduct* oder *idVendor*-Felder zu nutzten und damit sicher gestellt ist, dass nicht mehrere Hersteller die selbe *idProduct* verwenden, müssen die Adressbereiche der *idProduct* bei dem USB Implementers Forum, Inc. gekauft werden. Dazu gibt es zwei Möglichkeiten. Man kann entweder ein Mitglied der Forums werden oder für einen einmalig Betrag einen Adressraum erstehen, man darf dann jedoch nicht das offizielle USB-Logo verwenden. [9] Im folgenden werden die für dieses Dokument interessanten Felder weiter erläutert:

idVendor: Das *idVendor*-Feld wird von der USB Implementers Forum, Inc. festgelegt. Das Feld ist 2 Byte lang und ein Wert ist genau einem Hersteller zugeordnet. Ersteht ein Unternehmen einen *idVendor*-Wert, kann er solange er diesen *idVendor*-Wert nutzt, frei über das *idProduct*-Feld verfügen.

idProduct: Das *idProduct*-Feld wird von einem Unternehmen vergeben, welches einen Wert im idVendor-Feld gekauft hat. Es ist ebenfalls 2 Byte lang. Damit könnte ein Unternehmen bis zu 2^{16} verschiedene Produkte beschreiben.

4. Policies

Eine Policy ist ein Regelwerk, welches Rechte und Möglichkeiten von Benutzern auf einem IT-System beschreibt und eingrenzt. Es gibt verschiedene Arten Policies, im folgenden wir nur die Beschrieben, welche für den weiteren Verlauf der Arbeit relevant ist. Hier besteht eine Regel aus mehreren einzelnen Bestandteilen, welche entweder wahr oder falsch sind. Diese Bestandteile können per *und*- oder *oder*-Verknüpfungen zu einer Regel vereint werden. Abstrakt ist eine Police mit einem Regelwerk wie der Straßenverkehrsordnung zu vergleichen. Auch hier gibt es Vorgaben wer, wann und wo fahren oder parken darf. So wird zum Beispiel bei einem Durchfahrtsverbot, welches für Anlieger ausgeschlossen ist, folgende Regel festgelegt:

Regel-1: Die Durchfahrt ist für alle verboten

Regel-2: *oder* der Fahrer ist Anlieger

Bezeichnen wir in dem Beispiel den Ausgang *der Fahrer darf durch die Straße fahren* als *1* und den Ausgang *der Fahrer darf nicht durch die Straße fahren* als *0*, so wäre hier das Ergebnis

$$\alpha = Regel\text{-}1 \vee Regel\text{-}2$$

mit

$$Regel\text{-}1 = 0$$

. Somit ergeben sich daraus für die verschiedenen Fälle von *Regel*-2

$$\alpha = \begin{cases} 0 & \text{wenn Regel-1 gleich 0} \\ 1 & \text{wenn Regel-1 gleich 1} \end{cases}$$

Ähnliche Regeln können über Policies auf Rechner festgelegt werden. Hier wäre eine mögliche vergleichbare Regel im Bezug auf Speicherzugriffe

1. Der Zugriff auf diesen Ordner ist gesperrt
2. Außer der Benutzer hat die Kennung MaxMuster

Solche Regeln werden jedoch nicht nur für die Organisation von Speicherzugriffen sondern auch für das Sperren bestimmter Einstellungen oder mancher Geräte verwendet.

5. Umgehung der USB-Policies

5.1. Wie wird gefiltert

Die in diesem Dokument benutzten USB-Policies werden über die USB-Deskriptoren 3.2 definiert. Wollten wir etwa ein Gerät mit $idProduct = 0x01$ und $idVendor = 0x02$ freigeben aber alle sonstigen Geräte abweisen, so wäre folgende Regel möglich:

- Verbiete alle USB-Geräte
- Erlaube USB-Geräte mit
  - $idProduct = 0x01$
  - $idVendor = 0x02$

Die Regeln werden von oben nach unten gelesen, wobei spätere Regeln frühere überschreiben. Hier würden also zuerst alle USB-Geräte blockiert werden, außer das Gerät besitzt die $idProduct = 0x01$ und $idVendor = 0x02$. Dies scheint logisch, hat ein Gerät z.B. die $idProduct = 0x03$, so tritt die *Erlaube*-Regel nicht in Kraft und es bleibt die *Verbiete*-Regel bestehen. Meldet sich ein Gerät mit $idProduct = 0x01$ und $idVendor = 0x02$ an, so gilt zwar auch zunächst die *Verbiete*-Regel, jedoch trifft die *Erlaube*-Regel zu und überschreibt die *Verbiete*-Regel, sodass der Zugriff gewährt wird. Diese Zugriffe können gegebenenfalls noch um eine *Active-Directory-Gruppe* erweitert werden. Dies ist vor allem nützlich, wenn man nur bestimmten Benutzern die Möglichkeit geben will, auf USB-Geräte zu zu greifen. Wollten wir z.B. dem Benutzer „Alice“den Zugriff auf ein USB-Gerät mit der $idProduct = 0x01$ und der $idVendor = 0x02$ geben, so wäre die Regel:

- Verbiete alle USB-Geräte
- Ist $User = Alice$
- Erlaube USB-Geräte mit
  - $idProduct = 0x01$
  - $idVendor = 0x02$

5.2. Teensy

Das Teensy ist ein Platine bestehend aus einem 72 MHz MK20DX256VLH7 Cortex-M4 Prozessor, 256 kbytes Flash Speicher und 64 kbytes RAM. Zudem verfügt es über eine USB-Schnittstelle. Man kann also ein Programm auf dem Teensy ablegen und dieses wird ausgeführt, wenn man den USB-Stick einsteckt. So kann man beliebige Signalfolgen über USB an ein anderes Gerät schicken.

5.3. Konzept

Da die USB-Felder nicht durch Signaturen oder sonstige Möglichkeiten vor Manipulation geschützt sind, sollte es möglich sein, einen Teensy so zu programmieren, dass er sich als ein beliebiges Gerät ausgibt, also beliebige *idProduct*- und *idVendor*-Werte emuliert. Beschränkt eine USB-Policy den Zugriff auf ein bestimmtes Gerät, so könnte man dieses theoretisch mit dem Teensy nachahmen. Um dies umzusetzen wurden verwendet:

- Teensy 3.1 + USB-Kabel
- Arduino 1.0.5 (64bit) installiert unter *~/teensy/arduino-1.0.5*
- Teensyduino 1.19 (64bit)
- Kali-Linux als Testbetriebssystem (64bit)

Zur Analyse, mit welchen idVendor und idProduct-Werten sich der Teensy meldet, wurde mit dem Kommando „tail -f /var/log/syslog“das zentrale Logfile des Linuxsystems ausgelesen. Beim ersten einstecken ergab sich dabei folgende Meldung:

```
[...]
Oct 26 20:44:26 kali kernel: [12143.520149] usb 3-1: new full-speed USB device number 8 using uhci_hcd
Oct 26 20:44:26 kali kernel: [12143.689151] usb 3-1: New USB device found, idVendor=16c0, idProduct=0482
Oct 26 20:44:26 kali kernel: [12143.689155] usb 3-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
Oct 26 20:44:26 kali kernel: [12143.689158] usb 3-1: Product: Keyboard/Mouse/Joystick
Oct 26 20:44:26 kali kernel: [12143.689160] usb 3-1: Manufacturer: Teensyduino
Oct 26 20:44:26 kali kernel: [12143.689163] usb 3-1: SerialNumber: 413450
Oct 26 20:44:26 kali kernel: [12143.696631] input: Teensyduino Keyboard/Mouse/Joystick as /devices/pci0000
     :00/0000:00:1d.0/usb3/3-1/3-1:1.0/0003:16C0:0482.000F/input/input31
Oct 26 20:44:26 kali kernel: [12143.696732] hid-generic 0003:16C0:0482.000F: input,hidraw0: USB HID v1.11
     Keyboard [Teensyduino Keyboard/Mouse/Joystick] on usb-0000:00:1d.0-1/input0
Oct 26 20:44:26 kali kernel: [12143.702215] input: Teensyduino Keyboard/Mouse/Joystick as /devices/pci0000
     :00/0000:00:1d.0/usb3/3-1/3-1:1.1/0003:16C0:0482.0010/input/input32
Oct 26 20:44:26 kali kernel: [12143.702411] hid-generic 0003:16C0:0482.0010: input,hidraw1: USB HID v1.11 Mouse [
     Teensyduino Keyboard/Mouse/Joystick] on usb-0000:00:1d.0-1/input1
Oct 26 20:44:26 kali kernel: [12143.708320] hid-generic 0003:16C0:0482.0011: hidraw2: USB HID v1.11 Device [
     Teensyduino Keyboard/Mouse/Joystick] on usb-0000:00:1d.0-1/input2
Oct 26 20:44:26 kali kernel: [12143.714319] input: Teensyduino Keyboard/Mouse/Joystick as /devices/pci0000
     :00/0000:00:1d.0/usb3/3-1/3-1:1.3/0003:16C0:0482.0012/input/input33
Oct 26 20:44:26 kali kernel: [12143.714650] hid-generic 0003:16C0:0482.0012: input,hidraw3: USB HID v1.11
     Joystick [Teensyduino Keyboard/Mouse/Joystick] on usb-0000:00:1d.0-1/input3
[...]
```

Listing 5.1: tail -f syslog output

Die wichtigen Werte, also die *idVendor* gleich *16c0* und die *idProduct* gleich *0482* sind farblich hervorgehoben. Nun bearbeitet man unter *arduino-1.0.5/hardware/teensy/cores/teensy3/* liegende *usb-desc.h*, welche die notwendigen Informationen bei einer Neubeschreibung des Teensy bereit hält. Der relevante Abschnitt sowie die zu ändernden Werte sind wieder farblich hinterlegt.

```
#elif defined(USB_HID)
  #define VENDOR_ID             0xBEEF //was 0x16C0
  #define PRODUCT_ID            0xBEEF //was 0x0482
  #define MANUFACTURER_NAME     {'T','e','e','n','s','y','d','u','i','n','o'}
  #define MANUFACTURER_NAME_LEN 11
  #define PRODUCT_NAME          {'K','e','y','b','o','a','r','d','/','M','o','u','s','e','/','J','o','y','s','t',
        'i','c','k'}
  #define PRODUCT_NAME_LEN      23
  #define EP0_SIZE              64
  #define NUM_ENDPOINTS         5
  #define NUM_USB_BUFFERS       24
  #define NUM_INTERFACE         4
  #define SEREMU_INTERFACE      2       // Serial emulation
  #define SEREMU_TX_ENDPOINT    1
  #define SEREMU_TX_SIZE        64
  #define SEREMU_TX_INTERVAL    1
  #define SEREMU_RX_ENDPOINT    2
  #define SEREMU_RX_SIZE        32
  #define SEREMU_RX_INTERVAL    2
  #define KEYBOARD_INTERFACE    0       // Keyboard
```

Listing 5.2: Ausschnitt: usb_desc.h

Ändert man hier die markierten Werte und beschreibt den Teensy mittels der Arduino-Software neu, so werden diese Deskriptoren verwendet. Die Einstellungen hierfür können Sie aus der Grafik A.2 im Anhang entnehmen. Das Programm ist dabei entbehrlich, hier wurde eine an den Teensy 3.1 angepasste Version des *Blink*-Programms verwendet, welches im Anhang abgelegt ist. Kompiliert man das Programm nun und lädt es auf den Teensy, ergibt der „tail -f /var/log/syslog“-Befehl folgenden Output:

```
[...]
Oct 26 20:45:46 kali kernel: [12223.536191] usb 3-1: new full-speed USB device number 13 using uhci_hcd
Oct 26 20:45:46 kali kernel: [12223.705175] usb 3-1: New USB device found, idVendor=beef, idProduct=beef
Oct 26 20:45:46 kali kernel: [12223.705185] usb 3-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
Oct 26 20:45:46 kali kernel: [12223.705193] usb 3-1: Product: Keyboard/Mouse/Joystick
Oct 26 20:45:46 kali kernel: [12223.705199] usb 3-1: Manufacturer: Teensyduino
Oct 26 20:45:46 kali kernel: [12223.705205] usb 3-1: SerialNumber: 413450
Oct 26 20:45:46 kali kernel: [12223.713431] input: Teensyduino Keyboard/Mouse/Joystick as /devices/pci0000
     :00/0000:00:1d.0/usb3/3-1/3-1:1.0/0003:BEEF:BEEF.001D/input/input40
Oct 26 20:45:46 kali kernel: [12223.713679] hid-generic 0003:BEEF:BEEF.001D: input,hidraw0: USB HID v1.11
     Keyboard [Teensyduino Keyboard/Mouse/Joystick] on usb-0000:00:1d.0-1/input0
Oct 26 20:45:46 kali kernel: [12223.720321] input: Teensyduino Keyboard/Mouse/Joystick as /devices/pci0000
     :00/0000:00:1d.0/usb3/3-1/3-1:1.1/0003:BEEF:BEEF.001E/input/input41
Oct 26 20:45:46 kali kernel: [12223.720739] hid-generic 0003:BEEF:BEEF.001E: input,hidraw1: USB HID v1.11 Mouse [
     Teensyduino Keyboard/Mouse/Joystick] on usb-0000:00:1d.0-1/input1
Oct 26 20:45:46 kali kernel: [12223.726334] hid-generic 0003:BEEF:BEEF.001F: hidraw2: USB HID v1.11 Device [
     Teensyduino Keyboard/Mouse/Joystick] on usb-0000:00:1d.0-1/input2
Oct 26 20:45:46 kali kernel: [12223.732342] input: Teensyduino Keyboard/Mouse/Joystick as /devices/pci0000
     :00/0000:00:1d.0/usb3/3-1/3-1:1.3/0003:BEEF:BEEF.0020/input/input42
Oct 26 20:45:46 kali kernel: [12223.732662] hid-generic 0003:BEEF:BEEF.0020: input,hidraw3: USB HID v1.11
     Joystick [Teensyduino Keyboard/Mouse/Joystick] on usb-0000:00:1d.0-1/input3
[...]
```

Listing 5.3: tail -f syslog output 2

Wie man den farblich hervorgehoben Stellen sehen kann, meldet sich der Teensy nun mit geänderten Deskriptoren.

5.4. Proof of Concept

Die Policy der virtuelle Umgebung ist so eingestellt, dass nur ein bestimmte Baureihe eines USB-Laufwerks an die virtuelle Umgebung durchgestellt wird.

TODO Bild/Log

Wird versucht ein Gerät mit abweichenden *idVendor* und *idProduct*-Werten zu verbinden, so wird diese abgelehnt und getrennt.

TODO Bild/Log

Werden die Deskriptoren des Teensies auf die des USB-Laufwerks geändert und angesteckt, so wird das Gerät durch gestellt.

TODO Bild/Log

6. Fazit und Gegenmaßnahmen

Da der USB-Standard keine Möglichkeit bietet Geräte fehlerfrei, zum Beispiel über Signaturen, zu identifizieren, kann diese Ebene nicht als effektive Schutzmaßnahme einstufen und muss die Gefahren direkt eindämmen. Jedoch ist dies im Bezug auf die Exploits auf Treiberebene sehr schwer. Die einfachste und sicherste Methode wäre, die Benutzung von USB-Ports in einem Unternehmen per Richtlinie zu verbieten und die Ports eventuell sogar noch physikalisch zu versiegeln. Hier hätte man natürlich den Nachteil, dass USB-Geräte nicht mehr direkt genutzt werden könnten. Als Lösung für dieses Problem, könnte man eine Art Schleusen-System für USB-Geräte aufgebaut werden. So könnte man, wenn ein USB-Stick an die Firma geschickt wird, dieser in dem Terminal-Server eingebunden und die Daten an den gewünschten Empfänger weiterreicht werden. Die Vorteile sind hier, dass falls Viren auf dem USB-Stick enthalten sind, diese vorher am Terminalserver sowie bei der Netzwerkübertragung von mehreren verschiedenen Virenscanner überprüft sowie heuristischen Analysten unterworfen werden könnten. Ebenso würde bei einem manipulierten USB-Stick nicht der Rechner des Mitarbeiters sondern nur der Schleusen-PC infiziert. Trifft man hier entsprechende Sicherheitsmaßnahmen wie die Platzierung des Schleusen-PCs in der demilitarisierten Zone und einen regelmäßigen Tausch oder Neuinstallation des Schleusenrechners, kann man das Risiko relativ gering halten. Der Mitarbeiter würde in diesem Fall nur die Dateien bekommen und wäre von der Manipulation auf Treiberebene nicht betroffen. Zudem sind die im Vorfeld getroffenen Sicherheitsmaßnahmen für den Mitarbeiter transparent.

A. Appendix

A.1. Quellcode

Listing A.1: Blink_2.ino

```
/* LED Blink, Teensyduino Tutorial #1
   http://www.pjrc.com/teensy/tutorial.html

   This example code is in the public domain.
*/

// Teensy 2.0 has the LED on pin 11
// Teensy++ 2.0 has the LED on pin 6
// Teensy 3.0 has the LED on pin 13
const int ledPin = 13;

// the setup() method runs once, when the sketch starts

void setup() {
  // initialize the digital pin as an output.
  pinMode(ledPin, OUTPUT);
}

// the loop() methor runs over and over again,
// as long as the board has power

void loop() {
  digitalWrite(ledPin, HIGH);   // set the LED on
  delay(1000);                  // wait for a second
  digitalWrite(ledPin, LOW);    // set the LED off
  delay(1000);                  // wait for a second
}
```

A.2. Ergänzende Grafiken

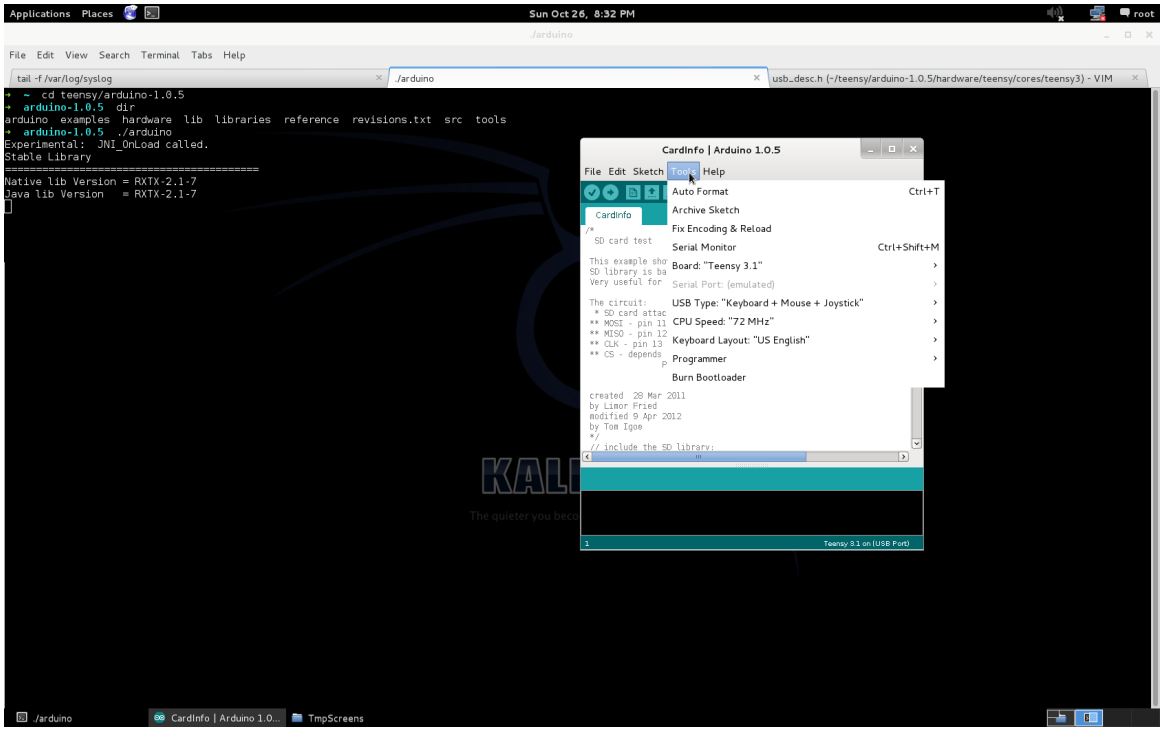

Abbildung A.1.: Einstellungen Arduino

A.3. Quellcode Grafiken

Listing A.2: Ablaufdiagramm 2.1

```
\begin{figure}[htbp]
\begin{tikzpicture}[
        scale=1,
        line width=0.25mm,
        every node/.style={
                scale=1,
                text=THIblue},
        align=center,
        node distance=4cm,
        comp/.style={
                fill=white,
                rectangle,
                draw,
                minimum size=2.5cm},
        driver/.style={
                fill=white,
```

```
17              rectangle,
18              draw,
19              yshift=2cm,
20              xshift=-1cm},
21      device/.style={
22              fill=white,
23              rectangle,
24              draw},
25      policie/.style={
26              fill=white,
27              rectangle,
28              draw,
29              yshift=-2cm,
30              xshift=-1.5cm}
31      ]
32
33      \node[comp] (thinclient) at (0,0){Thinclient};
34      \node[driver] (thinclientUSB) [below right of=thinclient] {
           USB-Treiber};
35      \node[device] (USBDevice) [right of=thinclientUSB, xshift=-1
           cm] {USB-Device};
36
37      \node[comp] (hypervisor) [above of=thinclient] {Hypervisor};
38      \node[driver] (hypervisorUSB) [below right of=hypervisor] {
           USB-Treiber};
39      \node[policie] (hypervisorPol) [above right of=hypervisor] {
           Policie};
40
41      \node[comp] (VM) [above of=hypervisor] {VM};
42      \node[driver] (VMUSB) [below right of=VM] {USB-Treiber};
43      \node[policie] (VMPol) [above right of=VM] {Policie};
44
45      \draw[->]
46              (USBDevice) --
47                      node[above]{1.}
48                              (thinclientUSB);
49
50      \draw[<->]
51              (thinclientUSB) --
52                      node[right]{2.}
53                              ++(0,-1) -| (USBDevice);
54
55      \draw[->]
56              (thinclient.125) --
57                      node[left]{3.}
58                              (hypervisor.235);
59
60      \draw[<->]
61              (hypervisor.10) --
62                      node[above]{4.}
63                              ++(2.5,0) |- (hypervisorPol);
64
```

```
65          \draw[->]
66                  (hypervisor.270) --
67                          node[left]{5.}
68                                  (thinclient.90);
69
70          \draw[->]
71                  (thinclient.55) --
72                          node[left]{6.}
73                                  ++(0,1.25)-|(hypervisorUSB);
74
75          \draw[->]
76                  (hypervisor) --
77                          node[left]{7.}
78                                  ++(0,2.5)-|(VMUSB);
79
80          \draw[<->]
81                  (VM.10) --
82                          node[above]{8.}
83                                  ++(2.5,0) |- (VMPol);
84
85  \end{tikzpicture}
86  \caption{Ablaufübersicht}
87  \label{fig:Ablauf}
88  \end{figure}
```

Listing A.3: USB Deskriptoren 3.2

```
1  \begin{wrapfigure}{l}{0pt}
2  \begin{tikzpicture}[scale=1, text=THIblue]
3          \draw (0,0) rectangle (0.5,0.5);
4          \draw (0.25, 0.25) node {1};
5          \draw (0.5, 0.25) node[right]{bLength};
6
7          \draw (0,0.5) rectangle (0.5,0.5);
8          \draw (0.25, 0.75) node {1};
9          \draw (0.5, 0.75) node[right]{bDescriptorType};
10
11         \draw (0,1) rectangle (0.5,1);
12         \draw (0.25,1.5) node {2};
13         \draw (0.5,1.5) node[right]{bcdUSB};
14
15         \draw (0,2) rectangle (0.5,0.5);
16         \draw (0.25,2.25) node {1};
17         \draw (0.5,2.25) node[right]{bDeviceClass};
18
19         \draw (0,2.5) rectangle (0.5,0.5);
20         \draw (0.25,2.75) node {1};
21         \draw (0.5,2.75) node[right]{bDeviceSubClass};
22
23         \draw (0,3) rectangle (0.5,0.5);
24         \draw (0.25,3.25) node {1};
25         \draw (0.5,3.25) node[right]{bDeviceProtocol};
```

```
26
27          \draw (0,3.5) rectangle (0.5,0.5);
28          \draw (0.25,3.75) node {1};
29          \draw (0.5,3.75) node[right]{bMaxPacketSize};
30
31          \draw (0,4) rectangle (0.5,1);
32          \draw (0.25,4.5) node {2};
33          \draw (0.5,4.5) node[right]{idVendor};
34
35          \draw (0,5) rectangle (0.5,1);
36          \draw (0.25,5.5) node {2};
37          \draw (0.5,5.5) node[right]{idProduct};
38
39          \draw (0,6) rectangle (0.5,1);
40          \draw (0.25,6.5) node {2};
41          \draw (0.5,6.5) node[right]{bcdDevice};
42
43          \draw (0,7) rectangle (0.5,0.5);
44          \draw (0.25,7.25) node {1};
45          \draw (0.5,7.25) node[right]{iManufacturer};
46
47          \draw (0,7.5) rectangle (0.5,0.5);
48          \draw (0.25,7.75) node {1};
49          \draw (0.5,7.75) node[right]{iProduct};
50
51          \draw (0,8) rectangle (0.5,0.5);
52          \draw (0.25,8.25) node {1};
53          \draw (0.5,8.25) node[right]{iSerialNumber};
54
55
56          \draw (0,8.5) rectangle (0.5,0.5);
57          \draw (0.25,8.75) node {1};
58          \draw (0.5,8.75) node[right]{bNumConfigurations};
59
60          \draw (0,9) rectangle (0.5,0.5);
61
62          \draw (0.25, 9) node[rotate=90, right] {Bytes};
63 \end{tikzpicture}
64 \label{fig:usbDeskriptoren}
65 \end{wrapfigure}
```

Literaturverzeichnis

[1] *AB Acquisition LLC Confirms Incident Involving Payment Card Data Processing.* http://www.jewelosco.com/2014/08/ab-acquisition-llc-confirms-incident-involving-payment-card-data-processing/, Abruf: November 1, 2014

[2] *Heartbleed.* https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160, Abruf: Oktober 29, 2014

[3] *Home Depot probes possible hack, could be larger than Target breach.* http://www.denverpost.com/business/ci_26453916/data-stolen-from-11-colorado-goodwill-stores-home, Abruf: November 1, 2014

[4] *McAffee.* http://csis.org/files/attachments/140609_McAfee_PDF.pdf, Abruf: Oktober 29, 2014

[5] *Shellshock.* https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271, Abruf: Oktober 29, 2014

[6] *Snowden.* https://www.theguardian.com/us-news/edward-snowden, Abruf: Oktober 29, 2014

[7] *USB 2.0 Standart.* http://www.usb.org/developers/docs/usb20_docs/#usb20spec, Abruf: Oktober 29, 2014

[8] *USB Implementers Forum, Inc.* http://www.usb.org/about, Abruf: Oktober 29, 2014

[9] *USB Vendor.* http://www.usb.org/developers/vendor/, Abruf: Oktober 29, 2014

[10] Stamp, Mark: *Information Security: Principles And Practice.* John Wiley & Sons, Inc., Hoboken, New Jersey, 2006. – S. 281 Malware